Examiner's Amendment and Statement of Reasons for Allowance

1. This action is responsive to Applicant's amendment filed May 13, 2009.

Examiner's Amendment

2. An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

Authorization for this examiner's amendment was given in a telephone interview with Mr. Ted Liu, Registration Number 60039, on June 9, 2009 for obviating any potential 101 issues and put the claims in condition for allowance.

The application has been amended as follows:

Specification:

Please replace paragraphs **[0003] – [0005]; and [0027]-[0028]**, with the new paragraphs **[0003] – [0005]; and [0027]- [0028]** as below.

[0003] The current invention relates generally to XA interactions in [[Java]] JAVA® Transaction API, and more particularly to parallel transaction processing in XA interactions in [[Java]] JAVA® Transaction API.

[0004] [[Java]] JAVA® standards for web services are constantly being developed. Concurrently, businesses are building important applications on top of web services infrastructures, such as that available in WebLogic Server from BEA Systems of San Jose, CA. As these applications evolve, they become more complex with more operations to perform. In many implementations, a server may perform transactions involving multiple resources. Transactional guarantees when making updates to the resources are important and should be performed as quickly as possible.

[0005] The WebLogic Server (WLS) Transaction Manager (TM) provides a full-featured two-phase commit transaction engine that implements the [[Java]] JAVA® Transaction API (JTA) specification. [[Java]] JAVA® 2 Enterprise Edition (J2EE) applications deployed on WLS make use of the TM and the JTA interfaces to effect atomic, persistent changes to data that are managed by one or more transactional resources. Transactional resources, such as RDBMS and messaging systems, support the XAResource interface as defined by the JTA specification. A representative server-resource system 100 is illustrated in FIG. 1.

In FIG. 1, an environment for a server computer 110 includes server 120, a first resource 130, and second resource 140. Server 120 includes transaction manager 150. Resources are accessed by an application to perform updates to persistent data. The transaction manager is used to ensure that the updates happen atomically with transactional guaranties. This process is described in more detail in "Open Group Distributed Transactions Processing" by Java Inc, incorporated by reference herein.

[0027] After the primary thread has processed the XA operations in step 560, the primary thread determines if the threads that received dispatches in step 560 have signaled the primary thread at step 570. In one embodiment, the primary thread determines if the respective threads have signaled the primary thread to indicate their respective commands have been processed. In one embodiment, operation remains at step 570 until all threads receiving dispatched commands have signaled the primary thread. In another embodiment, the primary thread will wait for a pre-determined period of time during which the threads receiving dispatches may signal the primary thread. In this embodiment, if the threads do not signal the primary thread within the time period, a time-out error is logged. If the TM is processing the prepare phase of the two phase commit protocol, then the transaction may be aborted. Once all threads processing dispatched requests have signaled the primary thread, the TM reports results of the XA operation at step 580. Operation then ends at step ~~595~~ <u>585</u>."

[0028] In one embodiment of the present invention, the signaling mechanism is based on a [[Java]] <u>JAVA®</u> class that maintains a counter of the number of

signal events anticipated. The primary thread will invoke a wait method that blocks the caller until the object is signaled the set number of times. The modified prepare/commit/rollback algorithms sets the expected number of signal events to be N-1. Each request that is dispatched to a thread for execution is given a reference to the synchronization object. Upon completing the XA operation the thread will invoke the object's signal method before terminating.

Claims:

1. (Currently Amended): A method for implementing a two-phase commit protocol, comprising:

associating, via a transaction manager, a plurality of resources in a transaction, wherein the plurality of resources are applied in the transaction using a prepare phase and a commit phase, wherein the prepare phase comprises a plurality of prepare operations, each of which is associated with one of the plurality of resources, and wherein the commit phase comprises a plurality of commit operations, each of which is also associated with one of the plurality of resources;

dispatching, via a thread pool manager, a second prepare operation from a first server thread to a second server thread, wherein the second prepare operation is associated with a second resource and the prepare phase, and the second prepare operation is executed by sending a preparation instruction from the second server thread to the second resource;

processing a first prepare operation by the first server thread in parallel to the second prepare operation being processed by the second server thread, wherein the first prepare operation is associated with a first resource and the prepare phase,

and the first prepare operation is executed by sending another prepare instruction from the first server thread to the first resource;

determining, via the thread pool manager, that the prepare phase is complete when both the first prepare operation and the second prepare operation are completed;

after determining that the prepare phase is complete, dispatching, via the thread pool manager, a second commit operation from the first server thread to a third server thread, wherein the second commit operation is associated with the second resource and the commit phase, and the second commit operation is executed by sending a commit instruction from the third server thread to the second resource;

processing a first commit operation by the first server thread in parallel to the second commit operation being processed by the third server thread, wherein the first commit operation is associated with the first resource and the commit phase, and the first commit operation is executed by sending another commit instruction from the first server thread to the first resource; and

determining, via the thread pool manager, that the commit phase is complete when both the first commit operation and the second commit operation are completed.

2. (Previously Presented): The method of claim 1 further comprising:
selecting an idle server thread to process the first prepare operation.

3. (Original): The method of claim 2, wherein selecting includes:
determining available server threads in a server.

4. (Currently Amended): The method of claim 3 wherein [[a]] the thread pool manager determines the available server threads in the server.

5. (Previously Presented): The method of claim 1 further comprising:

reporting results of the prepare phase to a log.

6. (Currently Amended): A method for processing a two-phase commit protocol operations, comprising:

associating, via a transaction manager, a plurality of resources in a transaction, wherein the plurality of resources are applied in the transaction using a prepare phase and a commit phase, wherein the prepare phase comprises a plurality of prepare operations, each of which is associated with one of the plurality of resources, and wherein the commit phase comprises a plurality of commit operations, each of which is also associated with one of the plurality of resources;

processing, via a thread pool manager, the plurality of prepare operations in a first server thread, wherein the processing of each prepare operation of the plurality of the prepare operations comprises:

dispatching the prepare operation to another server thread if the another thread is available; and

processing the prepare operation in the first server thread if no other server thread is available;

determining, via the thread pool manager, that the prepare phase is complete, when every one of the plurality of prepare operations completes;

after determining that the prepare phase is complete, processing, via the thread pool manager, the plurality of commit operations in a first server thread, wherein the processing of each commit operation of the plurality of the commit operations comprises:

dispatching the commit operation to another server thread if another server thread is determined to be available;

processing the commit operation in the first server thread if no other server thread is available; and

determining, via the thread pool manager, that the commit phase is complete, when every one of the plurality of commit operations completes.

7. (Canceled).

8. (Currently Amended): The method of claim 6 wherein [[a]] the thread pool manager determines the available server threads in the server.

9. (Canceled).

10. (Currently Amended): The method of claim 6 further comprising:

reporting results of the [[the]] plurality of prepare operations associated with the prepare phase to a log.

11. (Previously Presented): The method of claim 1, wherein a dedicated thread pool is used for parallel transaction operations.

12. (Currently Amended): The method of claim 1, wherein a transaction manager implements [[Java]] JAVA® Transaction API.

13. (Previously Presented w): The method of claim 1, wherein the first resource is an XA resource.

14. (Currently Amended): The method of claim 6, wherein each prepare operation of [[N-1]] the plurality of the prepare operations is associated with an XA resource.

15. (Previously Presented): The method of claim 6, wherein a dedicated thread pool is used for parallel transaction operations.

16. (Currently Amended): The method of claim 6, wherein [[a]] the transaction manager implements [[Java]] JAVA® Transaction API.

17. (Currently Amended): A system, comprising:

one or more processors;

a transaction manager, wherein the transaction manager associates a plurality of resources in a transaction, wherein the plurality of resources are applied in the transaction using a prepare phase and a commit phase, wherein the prepare phase comprises a plurality of prepare operations, each of which is associated with one of the plurality of resources, and wherein the commit phase comprises a plurality of commit operations, each of which is also associated with one of the plurality of resources; and

a dedicated thread pool, on a server machine, for parallel transaction operations, including:

a first server thread, wherein the first server thread processes a first prepare operation, and wherein the first prepare operation is associated with a first resource and [[a]] the prepare phase;

a second server thread, wherein the first server thread dispatches a second prepare operation to the second server thread, wherein the second prepare operation is associated with a second resource and the prepare phase and processed by the second server thread in parallel to the first prepare operation being processed by the first server thread; and

a third server thread, wherein after the transaction manager determines that the prepare phase is complete, the first server thread processes a first commit operation associated with the first resource, and the first server thread dispatches to the third server thread a second commit operation associated with the second resource, wherein the second commit operation is processed by the third server thread in parallel to the first commit operation being processed by the first server thread.

18. (Previously Presented): The system of claim 17 further comprising:

a thread pool manager, wherein the thread pool manager determines available server threads.

19. (Previously Presented): The system of claim 17 further comprising:

a transaction log, wherein the transaction log records results of the prepare phase and the commit phase.

20. (Previously Presented): The method of claim 1, wherein all of the prepare operations and all of the commit operations are part of a single transaction.

21. (Previously Presented): The method of claim 6, wherein all of the prepare operations and all of the commit operations are part of a single transaction.

-- The End --

Examiner's Statement of Reason(s) for Allowance

3. Claims 1-6, 8, 10-21 are allowed.

4. The following is an examiner's statement of reasons for allowance:

The prior arts of record: **Lampson** et al., teaches a two-phase commit protocol for a distributed transaction processing system employs the presumed-commit configuration, with the exception that the new presumed-commit protocol coordinator needs to force-write only a "commit" log record for committed transactions, not the previous force writing of two log records. In order to provide information needed to allow the coordinator to answer inquiries from subordinate processes following a crash or loss of communications, a technique for circumscribing the set of indeterminate transactions is employed. The transactions are numbered in increasing order, identified by a transaction ID. **Motomura**, teaches a parallel processor system executing a program consisted of a plurality of threads in parallel per threads, includes thread generating portion for managing three states of executing state, executable state and waiting state as states of the threads and generating other thread in the executable state by fork operation from the threads in the executing state on predetermined processor, thread execution control portion for making the first thread in the executable state to be executed on the processor, providing guarantee for data dependency between a plurality of first threads in the executing states on the processors, executing a second thread in the executable state in place of the first threads when the first thread in the executing state enters into waiting state with interrupting execution, and re-executing the first thread in the waiting state after termination of execution of the second thread. **Doolittle** et al., teaches a method when a server of the computing environment receives a request to be processed and that request is waiting on a response from a

client of the computing environment, the set of eligible thread pools for the response is dynamically altered. This dynamic altering allows the response to be serviced by a thread pool different from the thread pool servicing the request, thereby avoiding a deadlock situation. New arts made of record: US 2003/0004774 by **Greene** et al., teaches a two-phase commit process implemented by the transaction manager is described. After the client makes a "commit" call to the transaction manager via the TXN object, the transaction manager makes a "prepare" call to all participants (resource managers) that have joined the transaction with the client. If all participants accept the "prepare" call, then the transaction manager issues a "commit" call to all participants. Here again, the participant may accept the call or abort the transaction. Should every participant acknowledge that it accepts the commit, the participants then perform the requested transaction and the transaction manager notifies the client that the transaction has been accomplished, and the process ends. US 2004/0107381 by **Bomfim** et al., teaches a high performance transaction storage and retrieval system supports an enterprise application requiring high volume transaction processing using commodity computers meeting business processing time budget requirements. A synchronization point or synchpoint refers to a database commit point or a data/file aggregate point in this document. The synchpoint follows a streamlined 2-phase protocol, where a "prepare to commit" (phase 1) signal is transmitted by the synchpoint coordinator to all synchpoint partners; a positive response (or vote) must be received by the coordinator from all partners before the actual "commit" (phase 2) command is issued. Any negative vote at any point prevents a successful commit and all partners must in this case back out the work done during the cycle. And US Patent No. 7,290,056 by **McLaughlin** et al.,

teaches a system for executing distributed transactions. A participant and a coordinator cooperate to execute a distributed transaction, the distributed transaction including a transaction executed by the participant. To manage the transaction, the coordinator and the participant communicate over a network using, for example, a stateless protocol. The transaction manager uses the TX protocol to begin and end transactions and the TX protocol manages transactions through three services--tx_begin, tx_commit and tx_rollback. The X/Open client sends a tx_begin to transmit programming instructions, tx_commit to commit a transaction and tx_rollback to terminate a partially complete transaction (e.g., a transaction prior to a tx_commit). The interface between the transaction and resource managers is the XA protocol and the XA protocol has two services--xa_prepare and xa_commit, which are triggered by receipt of a tx_commit from a X/Open client. The transaction manager uses a two-phase commit protocol where the xa_prepare service to advise the resource manager to effect a pre-commit and the xa_commit to effect a commit. And US Patent No. 6,298,370 by **Tang** et al., teaches A process of operating a computer system. The computer system has a storage holding an operating system (OS) and an application program (APP.exe), a first processor having an instruction set, and a second processor having a different instruction set. The process includes steps of 1) running at least some of the operating system (OS) on the first processor so that the first processor sets up for at least part of the application program at run time at least one second processor object (VSP OBJECT 1); and 2) concurrently running the second processor to access the second processor object (VSP OBJECT1) and thereby determine operations for the second processor to access second processor instructions for said part of the application program (APP.exe) and data to be processed according to

said second processor instructions, and running the second processor to process the data according to said second processor instructions. However, none of them, taken alone or in combination, teaches the features in such a manner as recited in independent claims 1, 6, and 17.

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

5. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Chih-Ching Chow whose telephone number is 571-272-3693. The examiner can normally be reached on 8:00am - 4:30pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Wei Zhen can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Chih-Ching Chow/
Examiner, Art Unit 2191
6/15/2009
/Wei Y Zhen/
Supervisory Patent Examiner, Art Unit 2191